Computational Linguistics: Crash Course on Prolog

Raffaella Bernardi

KRDB, Free University of Bozen-Bolzano

P.zza Domenicani, Room: 2.28, e-mail: bernardi@inf.unibz.it

Contents

1. Introduction

Today we will look at how to use PROLOG to store information, namely to store a knowledge base of facts and how to ask queries.

2. Knowledge Base

```
wizard(harry).
wizard(ron).
wizard(hermione).
muggle(uncle_vernon).
muggle(aunt_petunia).
chases(crookshanks, scabbars).
```

Given this KB, you can ask for instance the following queries

```
?- wizard(harry).
yes
?- muggle(harry).
no
?- witch(hermione).
ERROR: Undefined procedure: witch/1

?- chases(X,Y).
X = crookshanks
```

```
Y = scabbars ;
no
?- chases(X,X).
no
```

3. A bit of syntax: atoms and variables

Atoms

- All terms that consist of letters, numbers, and the underscore and start with a **non-capital letter** are atoms: `harry`, `uncle_vernon`, `ritaSkeeter`, `nimbus200` ...
- All terms that are enclosed in single quotes are atoms: 'Professor Dumbledore', '(@ *+ ', ...
- Certain special symbols are also atoms: `+`, `,` , ...

Variables

- All terms that consist of letters, numbers, and the underscore and start with a **capital letter** or an underscore are variables: `X`, `Hermione`, `_ron` ...
- _ is an anonymous variable: two occurrences of _ are different variables.

4. A bit of syntax: complex terms

Complex terms

- Complex terms are of the form: $functor(argument, ..., argument)$.
- Functors have to be atoms.
- Arguments can be any kind of Prolog term, e.g., complex terms. `likes(ron,hermio` `likes(harry,X)` but also `f(a,b,g(h(a)),c)`, ...

5. Facts and Queries

Facts Facts are complex terms which are followed by a full stop.

```
wizard(hermione).
muggle(uncle vernon).
chases(crookshanks,scabbars).
```

5. Facts and Queries

Facts Facts are complex terms which are followed by a full stop.

```
wizard(hermione).
muggle(uncle vernon).
chases(crookshanks,scabbars).
```

Queries Queries are also complex terms which are followed by a full stop.

? - `wizard(hermione).`

where, ? - is the prompt provided by the Prolog Interpreter and `wizard(hermione).` is the query.

6. kb2: a knowledge base of facts and rules

```
eating(dudley).
happy(aunt_petunia) :- happy(dudley).
happy(uncle_vernon) :- happy(dudley), unhappy(harry).
happy(dudley) :- kicking(dudley,harry).
happy(dudley) :- eating(dudley).
```

where,

- `:-` stands for "if ... then ...": If `happy(dudley)` is true, then `happy(aunt petunia)` is true.

6. kb2: a knowledge base of facts and rules

```
eating(dudley).
happy(aunt_petunia) :- happy(dudley).
happy(uncle_vernon) :- happy(dudley), unhappy(harry).
happy(dudley) :- kicking(dudley,harry).
happy(dudley) :- eating(dudley).
```

where,

- `:-` stands for "if ... then ...": If `happy(dudley)` is true, then `happy(aunt petunia)` is true.
- `,` stands for "and": If `happy(dudley)` is true and `unhappy(harry)` is true, then `happy(uncle vernon)` is true.

6. kb2: a knowledge base of facts and rules

```
eating(dudley).
happy(aunt_petunia) :- happy(dudley).
happy(uncle_vernon) :- happy(dudley), unhappy(harry).
happy(dudley) :- kicking(dudley,harry).
happy(dudley) :- eating(dudley).
```

where,

- :- stands for "if ... then ...": If `happy(dudley)` is true, then `happy(aunt petunia)` is true.
- , stands for "and": If `happy(dudley)` is true and `unhappy(harry)` is true, then `happy(uncle vernon)` is true.
- "or" is expressed by the last two facts. If `kicking(dudley,harry)` is true or if `eating(dudley)` is true, then `happy(dudley)` is true.

7. Queries to kb2

```
eating(dudley).
happy(aunt_petunia) :- happy(dudley).
happy(uncle_vernon) :- happy(dudley), unhappy(harry).
happy(dudley) :- kicking(dudley,harry).
happy(dudley) :- eating(dudley).
```

Some possible queries to kb2

```
?- happy(dudley).
yes
?- happy(aunt_petunia).
yes
?- happy(uncle_vernon).
no
?- happy(X).
X = aunt_petunia ;
X = dudley ;
no
```

8. A bit of syntax: Rules

Rules are of the form `Head :- Body.`

- Like facts and queries, they have to be followed by a full stop.
- Head is a complex term.
- Body is complex term or a sequence of complex terms separated by commas.

```
happy(aunt_petunia) :- happy(dudley).
happy(uncle_vernon) :- happy(dudley),
                       unhappy(harry).
```

9. Kb3: facts and rules containing variables

Let's take a knowledge base that defines 3 predicates: father/2, mother/2, and wizard/1.

```
father(albert,james).
father(james,harry).
mother(ruth,james).
mother(lili,harry).
wizard(lili).
wizard(ruth).
wizard(albert).

wizard(X) :- father(Y,X),
             wizard(Y),
             mother(Z,X),
             wizard(Z).
```

10. Rules

```
wizard(X) :- father(Y,X),
             wizard(Y),
             mother(Z,X),
             wizard(Z).
```

The rule says:

> For all X, Y, Z, if `father(Y,X)` is true and `wizard(Y)` is true and `mother(Z,X)` is true and `wizard(Z)` is true, then `wizard(X)` is true. I.e., for all X, if X's father and mother are wizards, then X is a wizard.

11. Queries to kb3

```
father(albert,james).
father(james,harry).
mother(ruth,james).
mother(lili,harry).
wizard(lili).
wizard(ruth).
wizard(albert).

wizard(X) :- father(Y,X),
             wizard(Y),
             mother(Z,X),
             wizard(Z).
```

Some possible queries to kb3

```
?- wizard(james).
yes
```

```
?- wizard(harry).
yes
?- wizard(X).
X = lili ;
X = ruth ;
X = albert ;
X = james ;
X = harry ;
no
?- wizard(X), mother(Y,X), wizard(Y).
X = james
Y = ruth ;
X = harry
Y = lili ;
no
```

12. Ancestors

Given the KB below, we want to define a predicate `grandparent_of(X,Y)` which is true if X is a grandparent of Y.

```
parent_of(paul,petunia).
parent_of(helen,petunia).
parent_of(paul,lili).
parent_of(helen,lili).
parent_of(albert,james).
parent_of(ruth,james).
parent_of(petunia,dudley).
parent_of(vernon,dudley).
parent_of(lili,harry).
parent_of(james,harry).
```

12. Ancestors

Given the KB below, we want to define a predicate `grandparent_of(X,Y)` which is true if X is a grandparent of Y.

```
parent_of(paul,petunia).
parent_of(helen,petunia).
parent_of(paul,lili).
parent_of(helen,lili).
parent_of(albert,james).
parent_of(ruth,james).
parent_of(petunia,dudley).
parent_of(vernon,dudley).
parent_of(lili,harry).
parent_of(james,harry).
```

```
grandparent_of(X,Y) :- parent_of(X,Z), parent_of(Z,Y).
```

13. Ancestors

Similarly,

```
greatgrandparent_of(X,Y) :- parent_of(X,Z),
                            parent_of(Z,A),
                            parent_of(A,Y).
greatgreatgrandparent_of(X,Y) :- parent_of(X,Z),
                                 parent_of(Z,A),
                                 parent_of(A,B),
                                 parent_of(B,Y).
```

13. Ancestors

Similarly,

```
greatgrandparent_of(X,Y) :- parent_of(X,Z),
                            parent_of(Z,A),
                            parent_of(A,Y).
greatgreatgrandparent_of(X,Y) :- parent_of(X,Z),
                                 parent_of(Z,A),
                                 parent_of(A,B),
                                 parent_of(B,Y).
```

This doesn't work for "ancestor of"; don't know 'how many parents we have to go back'.

14. Ancestor

```
ancestor_of(X,Y) :- parent_of(X,Y).
```

this says that People are ancestors of their children.

14. Ancestor

```
ancestor_of(X,Y) :- parent_of(X,Y).
```

this says that People are ancestors of their children.

then, we need to say that they are ancestors of **anybody** that their children may be ancestors of (i.e., of all the descendants of their children).

14. Ancestor

```
ancestor_of(X,Y) :- parent_of(X,Y).
```

this says that People are ancestors of their children.

then, we need to say that they are ancestors of **anybody** that their children may be ancestors of (i.e., of all the descendants of their children).

```
ancestor_of(X,Y) :- parent_of(X,Z), ancestor_of(Z,Y).
```

14. Ancestor

```
ancestor_of(X,Y) :- parent_of(X,Y).
```

this says that People are ancestors of their children.

then, we need to say that they are ancestors of **anybody** that their children may be ancestors of (i.e., of all the descendants of their children).

```
ancestor_of(X,Y) :- parent_of(X,Z), ancestor_of(Z,Y).
```

The presence of the same predicate in the head and the body of the rule indicates we have a **recursion**.

15. Lists

Intuitively: sequences or enumerations of things.

15. Lists

Intuitively: sequences or enumerations of things.

In Prolog: a special kind of data structure, i.e., special kinds of Prolog terms.

15. Lists

Intuitively: sequences or enumerations of things.

In Prolog: a special kind of data structure, i.e., special kinds of Prolog terms.

```
[] The empty list
[Head|Tail] is a list if
Head is a term (atom, variable, complex term) and Tail is a list.
```

15. Lists

Intuitively: sequences or enumerations of things.

In Prolog: a special kind of data structure, i.e., special kinds of Prolog terms.

```
[] The empty list
[Head|Tail] is a list if
Head is a term (atom, variable, complex term) and Tail is a list.
```

For instance,

```
[a,b,c] A list with elements a, b and c
[a|Tail] A list with the element a and the elements in the Tail
```

you can also find eg. `[a,b | [c,d]]` for the list `[a, b, c, d]`.

16. Concatenation

`concatenate/3`: a predicate for concatenating two lists. `concatenate(X,Y,Z)` should be true if Z is the concatenation of X and Y; for example, concatenating [a] with [b,c] yields [a,b,c].

16. Concatenation

`concatenate/3`: a predicate for concatenating two lists. `concatenate(X,Y,Z)` should be true if Z is the concatenation of X and Y; for example, concatenating [a] with [b,c] yields [a,b,c].

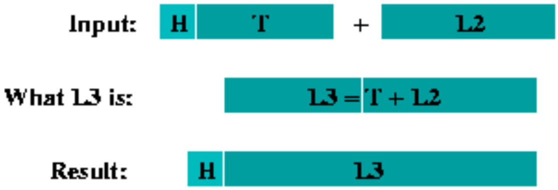
This predicate `concatenate(X,Y,Z)` is defined following the ideas below:

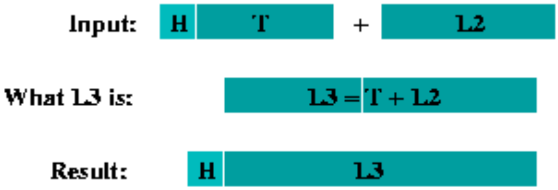
- if X is [], then Z=Y is the concatenation of X and Y.
- if X is the list `[H|T]` then `[H|T1]` is the concatenation of X and Y if T1 is the concatenation of T and Y.

Formally,

```
concatenate([],L,L).
concatenate([Head|Tail],L,[Head|NewTail]) :-
             concatenate(Tail,L,NewTail).
```

Remark, "append" is an alternative way of calling the predicate "concatenate".


Input:
H
T
+
L2
What L3 is:
L3 = T + L2
Result:
H
L3


Input:
H
T
+
L2
What L3 is:
L3 = T + L2
Result:
H
L3

17. Split List

concatenate (or append) can also be used in other ways. For example, to split lists into two parts.

17. Split List

concatenate (or append) can also be used in other ways. For example, to split lists into two parts.

```
?- append(X,Y,[a,b,c]).
X = []
Y = [a,b,c] ;

X = [a]
Y = [b,c] ;

X = [a,b]
Y = [c] ;

X = [a,b,c]
Y = [] ;

no
```

18. Conclusion

Now have fun using Prolog!!